Modeling Interactions

One of the underlying assumptions in many mathematical models in biology is that interacting systems are uniformly mixed. For example, in the case of modeling the growth of an organism with the logistic equation we assume that that the organism and available yeast is uniformly distributed at all times. One of the benefits of working with agent based modeling is that we can drop this assumption. Often interactions lead to a spatial distribution that is not uniform. In this lab we will use NetLogo to model the growth of different interacting organisms and then analyze the data in order to compare it with analytical models of population growth. In the process we will encounter some new NetLogo features, including breeds, behavior-space and shapes editor.

Breeds

Different types of turtles in NetLogo can be specified by defining breeds. Each breed can have its own attributes and can be controlled with breed-specific commands. You can also specify different shapes and sizes for each breed.

The first model we will explore will be a case of tetrahymena feeding on randomly floating pieces of yeast. We will use NetLogo's Behavior Space to examine how a variety of parameters influence the growth curves for the tetrahymena and will make quantitative comparisons with the predictions of the Logistic Model. We will then gradually make additions and enhancements to this model, to reflect a more realistic interaction and see how these changes alter the growth of the tetra.

Starting the Model

Before starting your program set the worldview window to have max-pxcor and max-pycor to 30 with patch size 8. Now click on the procedures tab and at the top of the page type the command

```
breed [tetra]
breed [yeast]
```

This defines two new types of turtles called tetra and yeast. Now create a setup procedure, which looks like the following:

```
to setup
   clear-all
   setup-tetra
   setup-yeast
end
```

and then define the two setup commands as follows:

```
to setup-tetra
  create-tetra tetra-number [
     set color white
     set shape "circle"
     setxy random-xcor random-ycor]
end

to setup-yeast
 create-yeast yeast-number [
     set color brown
```

```
      set shape "circle"
      setxy random-xcor random-ycor]
end
```

In order for the above code to be complete you need to specify the range of values for the variables `tetra-number` and `yeast-number` using sliders on the interface page. Let the values range from 0 to 1000 in steps of 1.

Shapes editor

In order to make the model more representative of what we might see under a microscope lets design our tetrahymena and yeast to have “realistic” shapes. Open up the Shapes Editor under the Tools menu. There are many predefined shapes and there are more that can be imported. However, lets design our own shape. Click on “new” and you will find yourself with a rather primitive graphics editing window. Draw a basic tetrahymena shape – you can add a few cillia and organelles if you wish, but don't get too fancy – it slows the program down. Save your shape with name tetra, and then create a new, smaller shape called yeast. There is no need to get too elaborate – some irregular polygon will do. Save this one as yeast, and then return to your setup procedure and change the shapes from “circle” to “tetra” and “yeast” as appropriate.

Movement

Now you are ready to get the tetrahymena and yeast to move. In this first model we will just have the breeds move randomly – with perhaps the tetrahymena moving faster than the yeast, since they have their own locomotion. Define a `go` command as follows:

```
to go
  ask yeast [ wiggle .01 ]
  ask tetra [ wiggle .1]
end
```

where `wiggle` is a command which must be defined. As written this command takes one input, which specifies how far the turtles move in each time step. To define commands with inputs you specify the input variable in square brackets after the command name. A suitable wiggle command is defined below. You can modify this as you wish.

```
to wiggle [amount]
      left random 90
      right random 90
      forward amount
end
```

Test your code thus far by adding setup and go buttons on the interface.

Eating and Reproduction

At the moment all that happens is tetrahymena and yeast move randomly across the environment. Now we need to program some mechanism for the tetrahymena to eat the yeast and then reproduce.

We will have tetrahymena only eat yeast if they are on the same patch as the yeast. After eating yeast they will gain some energy. When their energy surpasses some defined threshold they will divide in half. Add two new procedures to your `go` procedure:

```
ask tetra [eat-yeast]
ask tetra [reproduce]
```

where `eat-yeast` can be defined as follows

```
to eat-yeast
           if any? yeast-here [
                 ask one-of yeast-here [die]
                 set energy  energy + yeast-energy     ]
end
```

In order for the `eat-yeast` code to work we need to make a slider for `yeast-energy` on the interface. Have this range from 1 to 10, with default 5. We also need to specify the tetra energy variable. Add the following statement under the line defining the breeds

```
tetra-own [energy]
```

and then in the setup-tetra procedure add a line

```
set energy random 20
```

This gives a random initial energy of the tetrahymena, between 0 and 19.

Tetrahymena will reproduce if their energy reaches twice this amount. A suitable `reproduce` command might be

```
to reproduce
  if (energy >= 20) [
    set energy energy / 2
    hatch 1 [ wiggle 1 ] ]
end
```

Note: when a turtle calls the hatch command, the new turtles have all the same attributes as the parent. Consequently, we do not need to specify these attributes, unless we want offspring to differ in some way from their parents.

Run your simulation to see if it behaves appropriately. Play around with different values of the parameters. Add a line at the bottom of the `go` command to get the program to stop when all the yeast is consumed. Something like this will do.

```
if (count yeast = 0 ) [stop]
```

Plotting your data

Add a labeled plot and a monitor to your interface showing the population of tetrahymena over time. You may want to glance at the first lab you did to see how best to do this.

Behavior Space.

After playing with the parameters in the model, you should notice that the growth pattern seems to be logistic (or sigmoid). In order to do more with our model than watch it we need to be able to analyze the data that is generated and compares it with either data in the field or some analytical model that we have derived. Lets compare this data with the predictions of the logistic model $\Delta N = rN(1 - N/K)$. To do this we will make use of NetLogo's Behavior Space. Behavior Space

is a way to do multiple runs of your model with different values of your parameters. The data is then outputted to a file with comma-separated values (csv), which can be read by a data analysis package such as Excel.

From the Tools menu select Behavior Space and choose NEW. Give the experiment a name. The second text box specifies the values of the parameter you want to test. For this experiment let's take the yeast-number to be 1000, keep the yeast-energy fixed at 10 and the initial number of tetra constant at 50. This is what you should type in this box

```
["yeast-number" 1000]
["yeast-energy" 10]
["tetra-number" 50]
```

In the next window you specify the number of runs. Let's run the experiment 10 times so that we can average runs to reduce statistical error.

The next text box specifies the data you want to analyze. In this case it should be

```
count tetra
```

The setup commands text box and go commands text box should have setup and go respectively as defaults. There is no need to change these unless you use different names for these procedures.
The stop condition text box specifies when a run is finished. Lets assume this is when the yeast runs out. Then the condition is

```
count yeast = 0
```

Leave the final commands text box empty and leave time limit set to 0. Then
click "ok" and run the experiment. You will find that it runs much more quickly if you uncheck "update view" and "update plots and monitors".

Save your data file as `data.csv`.

Analyzing the data in Excel

Open up the data file in Excel. The experimental runs, with number of tetra, N, should be neatly arranged in 10 columns for your analysis. Our purpose here is to compare the data with the predictions of the Logistic model. In particular, for the logistic model we expect that the per-capita growth rate $\Delta N/N$ will decrease linearly, with the y-intercept being the intrinsic growth rate *r* and the x-intercept being the carrying capacity *K*. In the column next to your data find the average of the 10 tetra populations at each time step. Then, in another column calculate $\Delta N/N$. Plot $\Delta N/N$ vs N. Now do a linear fit to your data. If a linear fit is a good fit, then obtain an estimate for *r*, the intrinsic growth rate. You might consider doing this again with different parameter values to see how they would affect *r.*

Assignment

In this lab you have been introduced to a number of new features of NetLogo in a model for the growth of tetrahymena. Now I would like you to implement enhancements so that your program will model more closely how a predator goes about catching its prey. My hope is that as you implement these suggestions you will learn more about modeling with NetLogo.

1. The first change to the model will be to arrange for tetra to die if they do not get enough yeast.
   (a) Make a slider called `metabolism` which ranges from 0 to 2 in steps of 0.1. Then change the model so that each tetrahymena loses an amount of energy equal to `metabolism` at each time step (ie in the `go` procedure).
   (b) Each time step `ask` tetrahymena with `energy` less than 0 to `die`.
2. In order that energy is not slowly drained away from the system create a few new yeast at random locations on the screen during each time step. The number you add at each time step could be determined by a new slider variable which you might call `yeast-growth-rate`. Choose an appropriate range so that tetrahymena have enough food to sustain them.
3. Another enhancement would be to take into account the fact that tetrahymena can sense the yeast and move towards it. Yeast produce a chemical which diffuses into the surrounding solution. The tetrahymena sense this chemical and turn to the direction where the concentration of the chemical is highest. Here is a sketch of what you might do to implement this idea:
   (a) Create a `patches-own` variable called `chemical` just below the line where you made the `tetra-own` variable called `energy`.
   (b) Now, during each time step ask the yeast to set the chemical value of the patch they are on to increase by one.
   (c) Now diffuse the chemical to neighboring patches using the `observer` command `diffuse chemical 1`. The number 1 refers to what fraction of the chemical should be diffused.
   (d) You should also allow the chemical to break down over time. Write a command that will reduce the chemical in each patch by 1% at each time step (or some other value that works well). You could make this percentage into a variable called `evaporation-rate` and add a slider to the interface.
   (e) To see the chemical concentration, ask the patches to color themselves according to the chemical concentration. You can do this with the `patch` command
   `set pcolor scale-color green chemical 0 10`
   This colors the patches with chemical concentration 0 black and patches with chemical concentration of 10 or above white. Patches with chemical concentration in between 0 and 10 are colored a shade of green.
   (f) Write a `seek-yeast` procedure for the tetrahymena. This procedure should ask the tetrahymena to determine which of the neighboring patches has the maximum chemical concentration (look up the `max-one-of` reporter in the user manual) and then `face` that patch. Add this new procedure in the `go` procedure.
4. Explore your model by changing the parameters. Find different combinations of parameter values that lead to approximately linear growth to the carrying capacity, to an oscillating population of tetra, a chaotic population pattern (you may have to adjust your `yeast-energy` variable to have higher upper bound to see this.)
5. Use Behavior Space to explore how the model behaves. For example, try setting `metabolism` and `yeast-growth-rate` to zero. Fix the yeast-number at a reasonable setting and then vary the `evaporation rate` to see what combination yields a growth curve that does not fit a logistic model well (ie the decrease in per capita growth rate is not linear.)
6. Submit your completed model to the dropbox by February 5th at 1:00 pm, using naming convention: `Lastname_firstname_Lab_2.nlogo`